

Sub-Workflow Interoperability, the Pros and Cons

Beth Plale¹, Yuan Luo¹, Kavitha Chandrasekar¹, Eran Chinthaka Withana²

¹Indiana University, Bloomington, IN

² NexTag Inc., San Mateo, CA

Workflow orchestration tools have gained recognition for their contributing role to scientific discovery. For example, a Taverna workflow designed to identify biological pathways implicated in the resistance to *Trypanosomiasis* was repurposed to study another parasite, *Trichuris muris*, by a change of data sets only. As workflow use proliferates, reproducibility inches closer to becoming reality. Reuse of a workflow, however, is guaranteed for only as long as the system on which it runs is operational. Further, the likelihood of a workflow designed for one workflow system running on another straightaway is low.

As workflow adoption proliferates, users will expect to use workflow snippets from multiple workflow systems, and it's not unreasonable for them to expect the snippets to compose into a single workflow. Why would they need to do this? First, workflow systems have become specialized. A system might be specialized to work with a back end Linux or Windows cluster. It may specialize in domain specific functionality (such as a unique set of tools for biologists or hydrologists). A system might be specialized to utilize a cloud platform as a back end, or provide special constructs for large-scale parallel execution of jobs. Second, since the likelihood of a workflow designed for one workflow system running on another is low, researchers will use the functionality when and where it exists. In the end, as workflow adoption proliferates, users will seek to use each system for what it is best at, creating the scenario where a portion of a workflow is run on one system and another portion on another system. As with social media sites, working within a single framework can be limiting. Interoperability between and across workflow systems, which is known as *subworkflow interoperability*, is possible. The WS-VLAM (Wibisono et al. 2007) system is a case in point. WS-VLAM has a common event service bus, the VL-e Workflow Bus that carries data and control flow to and from multiple remote workflow systems. Such an approach seems highly desirable, but at what cost? We explore that question fully in Plale et al. (2011) and summarize results here.

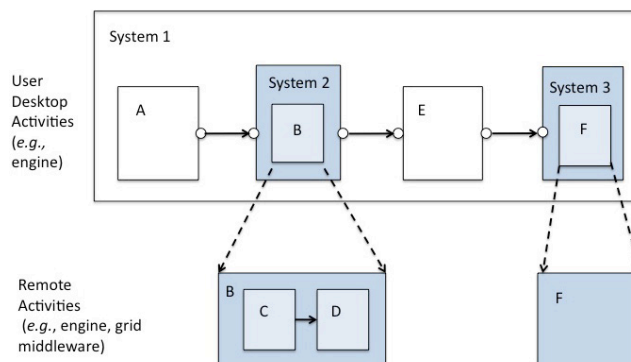


Fig. 1. Subworkflow interoperability shares workflows between systems.

Subworkflow interoperability takes several forms as illustrated in Fig. 1. System 1 is the top-level workflow system. For our study, it sits on a user desktop. System 1 orchestrates activities A, B, E, and F. Activity B is called from System 1 by instructing System 2 to execute the activities. While B is seen as a single activity by System 1, it is actually a workflow system that executes the workflow C->D. Activity F is activated through an invocation of System 3 by System 1. System 3 is a proxy for an activity that runs remotely such as on grid middleware. Other forms of subworkflow interactivity exist, but this set of cases is rich enough to work with. A system that can utilize local machine resources for simple execution and remote resources for more complex tasks is simpler in the simple case. Workflow system to workflow system interaction is complex, and that programming complexity should not hurt the simple case, thus enforcing the adage that what a user doesn't know should not hurt them. System 1 could be a user desktop workflow system like the Trident Scientific Workflow Workbench used in our study. Trident is easy to use in the sense that it is integrated with .NET and Workflow Foundation so writing a new activity is a straightforward coding effort.

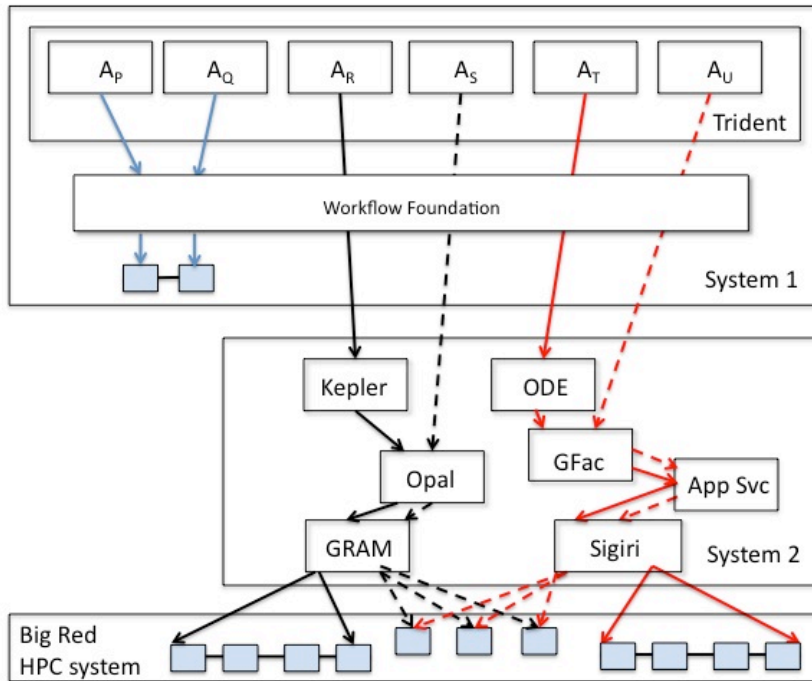


Fig. 2. Sub-workflow interoperability: local execution within System 1 is shown by activity A_P and A_Q invoking a local workflow. Activity A_R communicates with the Kepler remote engine to run a sub-workflow on Big Red (solid black lines). Activity A_S contacts grid services directly to invoke nodes individually (dotted black lines). Activity A_T invokes the ODE workflow engine to run a sub-workflow on Big Red (solid red lines). Activity A_U contacts grid services directly to invoke nodes individually (dotted red lines).

Based on the model in Fig. 1, we construct a set of tests to assess the pros and cons of constructing a system that has a top-level workflow system calling out to lower level workflow systems and remote resources. This system of systems we assemble for testing has similarities to WS-VLAM except we did

not integrate the event service bus that would provide a unified communication model. In our test system the top-level workflow orchestration system is Trident. The remote workflow systems used are the Kepler workflow system, and Apache ODE. Proxy execution (System 3) uses the GFac and Opal toolkits which proxy between a workflow environment and an arbitrary workflow graph node (such as legacy a Fortran code).

Architectural Organization

In our study we decomposed Fig. 1 into four high level components described as follows and illustrated in Fig. 2:

1. **Baseline execution environment:** The top-level workflow engine and local execution environment. Workflows are run locally. This shown as activities A_P and A_Q executing as part of a local workflow executed through Workflow Foundation.
2. **Remote workflow engine:** A_R and A_T are the remote workflow engine case. A_R invokes Kepler that contacts the Opal Toolkit to execute a sub-workflow on a supercomputer (called Big Red) through Globus GRAM. A_T invokes the Apache ODE workflow engine that contacts GFac to execute a sub-workflow on Big Red using the Sigiri resource manager.
3. **Remote grid/cloud middleware:** A_S and A_U illustrate activities which contact grid/cloud middleware directly. Activity A_S contacts Opal. Since there is no orchestration at the remote system, the remote grid/cloud services are capable of executing only one task of a workflow. Shown in the black dotted lines is the Globus GRAM resource manager executing one of the three services of a sub-workflow similarly activity A_U contacts GFac for execution of one of the three tasks pointed to by the dotted red lines.
4. **High performance computing resources:** large scale supercomputers or cloud resources.

The workflows used in the study cover four cases over two workflow patterns as follows:

Data Intensive, Sequential Workflow: consists of 5 sequential MD5 tasks, each of which applies a data movement operation and a cryptographic hash function (i.e., MD5) on a 1.5 GB file.

Data Intensive, Parallel Workflow: consists of 5 parallel executed MD5 tasks, each of which is the same as in Data Intensive, Sequential Workflow.

Compute Intensive, Sequential Workflow: consists of 5 sequential Linpack tasks, each of which is configured to carry out execution on a 5000 x 5000 matrix, using double precision floating point coefficients. The inputs and outputs are small, on the order of 5KB.

Compute Intensive, Parallel Workflow: consists of 5 parallel Linpack tasks, each of which is configured the same as the Compute Intensive, Sequential Workflow.

We identified several metrics for key overheads. First is the time penalty of using a second workflow engine. This is measured as the time difference to invoke workflow instance i on local machine versus time to invoke workflow through a remote workflow engine. Second, remote services and remote workflow engines are both remote, so what is the cost of using a remote workflow engine specifically? Third, workflow systems show varying latencies for complex workflows. The third and final measure captures that variability.

Summary of Results

We experimentally evaluated two models of remote execution for a handful of scenarios to illuminate the latencies and variability inherent in different approaches. We capture an overall measurement of sub-workflow overhead by running a workflow directly within the Trident workstation and comparing that against the same workflow executed by a secondary workflow engine. In order to have a fairer comparison, we ignore waiting time in the job queue in the remote case. For the compute-intensive workflows, the remote ODE workflow engine version had only 2.4% higher execution time than local execution. The remote grid service version using Gfac/Sigiri, on the other hand, had 15.5% higher execution time than local machine execution. This difference is due to overhead in invoking GFac, the service factory, which uses a model of dynamic web service creation. Kepler sub-workflow and Opal/GRAM grid services showed only 5% higher execution time than local machine execution. Hence, the difference in overhead can be attributed to the difference in architectural decisions in support of dynamic capabilities.

In the remote task approach, Trident has more activities to manage and track, whereas for the remote engine approach, the number of nodes in the Trident workflow is minimal. In sequential workflows, the remote workflow engine only adds 10% overhead to the overhead introduced by Trident. In the case of running parallel remote services in Trident, there is no concurrent execution, so the performance in Trident is up to 5 times worse than using sub-workflow engines which handle parallel execution.

Under sequential compute intensive workflows, ODE and Kepler stacks take approximately the same amount of time. Due largely to to Sigiri, the ODE stack keeps the submission overhead within a small range. Kepler uses Globus GRAM as its job submission middleware and GRAM takes a varying amount of time to get the job scheduled and executed in the compute resource. The variability of Kepler/Opal Stack is 14 times worse than ODE/Sigiri stack. During this experiment we also experienced job failures caused by different GRAM failures and had to re-run our experiments on multiple occasions.

There are softer aspects about which comparison can be made including locus of control, extensibility and architectural complexity.

Locus of Control. Remote grid/cloud services are capable of scheduling individual activities, since there is no orchestration at the remote system. In the case where the subworkflow system is invoked, the black-box nature of the subworkflow model relieves Trident of complex control because it hands that off to sub-workflow. The more minimal the control of Trident, the less the user has to understand the workflow.

Extensibility. Workflow engines are often bound to certain services to carry out tasks used during workflow execution which limits the functionality of the workflow engine. The ODE workflow engine, particularly through its instantiation in the OGCE tool suite is limited by its ability to add new workflow activities. New must be exposed as a Web service. Kepler and Trident workflow engines on the other hand are targeted to desktop executions and thus allow new functionality to be more easily incorporated into the system. With the actor model in Kepler and activity model in Trident, a user can program any functionality into the workflow, enabling a wide variety of functionality to be supported in the workflow. In experience gained in the research lab and in the classroom, Trident is easy to use. Programming new workflow activities requires writing a small C# activities. With its user-friendly interface and programming model, the tool could appeal to the scientist who is comfortable with a Windows platform and inclined to run on their local compute resources.

Architectural complexity. Architectural complexity captures the number of components, fragility of interfaces, etc. of a system. Architectural complexity is directly proportional to the availability and

reliability of the system. The higher the complexity, the lower its availability unless significant and costly measures are put in place for fault monitoring, replication, and recovery. The remote task approach (over remote engine approach) gives the user more control over the execution of the experiment. But with this control comes maintenance overhead and complexity, because the user has to manage all the components and their interactions. From our experience with the LEAD Science Gateway (2003-2010), grid middleware adds complexity to the architecture. When Trident is expected to interact directly with Sigiri and Opal, the workflow author will have to handle the complexities for each activity, including authentication mechanisms, fault tolerance and checkpointing. In the sub-workflow workflow engine approach, the user must provide perfect configuration parameters for the next workflow engine to function properly.

Conclusion

Grid middleware is responsible for a significant amount of overhead in a scientific workflow stack scheduling jobs into supercomputing resources. The job failures and the higher variation of overheads we experienced during our evaluation suggests that the instability and unpredictable behavior of grid middleware components have high impact on the scientific workflow systems that are using them. However efficient and optimized these workflow stacks are, the issues in middleware can make these workflow suites uncertain, if not unusable.

Similar to other workflow systems, Trident facilitates workflow runs in Windows based environments. But we think more improvements are necessary to make this toolkit more useable among the scientific research community. For example, Trident lacks the support for parallel execution constructs within it. Even though activities are picked up and scheduled in parallel, for a parallel workflow, they are executed sequentially.

The takeaway message of this study is that a user today can compose a workflow using a high level workflow engine whose subcomponents run on various lower level workflow systems or grid resources. The execution overheads are reasonable. The subworkflow system can even augment the capability of the higher-level workflow system. But this comes at the cost of additional system complexity. This complexity can be overcome through redundancy and resiliency measures, but often the latter are out of reach for open source research tools because of their substantially higher development and maintenance costs.

Further Reading

B. Plale et al. (2011) Strengths and Weaknesses of Sub-Workflow Interoperability. *Tech. Rep. TR700*, School of Informatics and Computing, Indiana University, Bloomington, Indiana

A. Wibisono et al. (2007) WS-VLAM: Towards a scalable workflow system on the Grid, *16th IEEE Int'l Symposium on High Performance Distributed Computing*